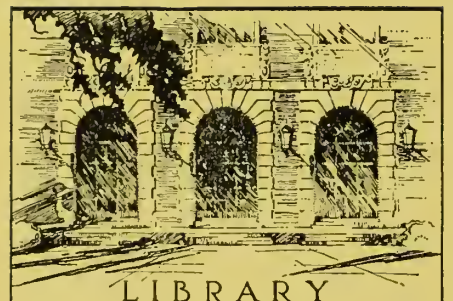

LIBRARY
OF THE
UNIVERSITY
OF ILLINOIS

510.84
I16r
no. 228-236
cop. 2

The person charging this material is responsible for its return to the library from which it was withdrawn on or before the **Latest Date** stamped below.

Theft, mutilation, and underlining of books are reasons for disciplinary action and may result in dismissal from the University.

UNIVERSITY OF ILLINOIS LIBRARY AT URBANA-CHAMPAIGN

MAY 31 1979
MAY 3 REC'D
JUN 7 1980
MAY 13 REC'D

MAY 04 1998

L161—O-1096

Digitized by the Internet Archive
in 2013

http://archive.org/details/websystempartiif232bond

Math

Report No. 232

COO-1018-1118

THE WEB SYSTEM
PART II* A Formal Description of the WEB Input Language

by

W. D. Bond
E. S. Davidson
B. H. McCormick

June 19, 1967

THE LIBRARY OF THE
AUG 30 1967
UNIVERSITY OF ILLINOIS

DEPARTMENT OF COMPUTER SCIENCE · UNIVERSITY OF ILLINOIS · URBANA, ILLINOIS

Report No. 232

THE WEB SYSTEM
Part II* A Formal Description of the WEB Input Language

by

W. D. Bond
E. S. Davidson
B. H. McCormick

June 19, 1967

Department of Computer Science
University of Illinois
Urbana, Illinois 61801

*Supported in part by Contract AT(11-1)-1018 with the U.S. Atomic Energy Commission and the Advanced Research Projects Agency

1. INTRODUCTION

It is the intent of this report to describe the input language for the WEB design automation system. An attempt has been made to keep the definition of the input language flexible. Accordingly, the input language should be applicable to design automation in widely differing technologies. Yet the language is sufficiently specific as to be presently useful for printed circuit construction.

The description of seemingly different types of input data needed for design automation has been kept syntactically similar wherever the problems of accurate specification were sufficiently analogous, even though the elements so described bear little or nothing in common. Thus, features may be found in the language for which the reader may have little immediate application. These features have been included either because they were required for some known application of the language, or because it was felt that future trends in computer technology might make these features desirable, or to preserve a certain symmetry in the definition of the language. In such cases precautions have been taken so that these features can be easily ignored by the user who has no current interest in using them.

It is not anticipated that the description of the WEB input language, as given here, will be used as an initial introduction to the language. Rather, a concise description is attempted for reference purposes. For an initial introduction, the reader is referred to

Part I of the WEB System: A Language to Describe the Interconnection and Packaging of Logical Units in a Digital Processor[1].

The WEB input consists of:

(1) A Logic Description, in which the logic of the system is described. Starting with the irreducible packagable logic units, block structure is used to define more complex system components in terms of previously defined ones.

(2) A Housing Description, in which connector geometries are described. Also using block structure format, more complex housings are defined in terms of arrays of pins.

(3) A Module Description, which might more generally be called a pluggable unit description, describes pluggable units (e.g., boards) in terms of their constituent logic units and the terminals to which they are connected.

(4) An Assignment Description, in which the packageable logic units are assigned to constituent logic units

---

1. J. H. Hazelhurst, B. H. McCormick and W. D. Bond, Department of Computer Science Report, University of Illinois, in progress. This report supercedes DCS File No. 625, Revised February 10, 1965.

of modules, and the modules (boards) are assigned actual locations in the housing.

As the main text of this paper is essentially a semantic description of the WEB language, the syntax found in section 6 should be read in conjunction with the text. Accordingly, throughout the text, underlined phrases correspond to major syntactic variables found in the syntactic description of WEB.

2. LOGIC DESCRIPTION

The logic description consists of a sequence of block definitions, each of which may be preceded by a title or drawing statement.

The title statement provides descriptive comment on the data being processed. Each title statement causes a new page to be started in the listing of the input description, and all subsequent pages will bear this title. The purpose of the drawing statement is to reference, by name, files kept elsewhere which contain drawings or other information pertinent to the data being processed.

A block definition defines a new block in terms of previously defined sub-blocks. If for some reason it is desirable to give a previously defined block a different name, a simple equivalence statement may be used rather than duplicate the entire definition.

Initially there are no predefined blocks. Hence one must first define all lowest level blocks by the use of unit descriptions. Accordingly, the unit description is given for each irreducible packageable unit.

After the units have been defined, higher level blocks are given block descriptions. Block descriptions differ from unit descriptions in that they contain listing and interconnection of component sub-blocks (units are lowest level blocks) and the reserved word BLOCK is used rather than UNIT.

A block description need not have a block identifier, in which case it is an "unnamed block" and cannot be used in the construction of higher order blocks.

New blocks can only be constructed by using previously defined named blocks: hence, all unit descriptions must have a unit identifier.

The terminal list is an ordered list of signal names. The choice of actual names used is immaterial except as these names have meaning to the user. A judicious choice of names, however, will enable the user to make opportune use of the iterative properties afforded by the FOR clause.

The signal names occurring in a terminal list have different connotations, depending on whether the terminal list occurs in the heading of a block description or as part of a component sub-block. The distinction is entirely anologous to that of formal (dummy) parameters versus actual (calling) parameters as used in sub-routines and macros. That is, let (actual) signal name α occur in the ith position of the terminal list of a component sub-block with block name β and let (formal) signal name γ occur in the ith position of the terminal list of the block description with block identifier β. Then each occurrence of the signal name γ in any sub-blocks of the block description of β is to be replaced by signal name α when the logic description is eventually expanded. Terminals renamed with the same signal name are electrically common.

Each signal name occurring in a block component of a named block, which does not occur in the terminal list of the block, is treated as a local actual signal name. Such a signal name is treated as a global actual signal name in the case of an unnamed block.

Embedded in the definition of signal element may be found simple identifiers and identifiers subscripted with one or more subscripts, as well as provision for coordination (identifiers with periods between them) for identifying parts of a tree containing either a single final node or an ordered set of final nodes, each of which ultimately becomes a signal element of the terminal list. It is for this latter purpose that a subscript may be a pair of integer expressions separated by a colon, e.g., "a:b", which means "from subscript a to subscript b 'inclusive'".

Signal group declarations are required if efficient use is to be made of the coordination facility, as well as to specify a sequence of signals by one signal name expression. The declaration is local to the block or unit in which it occurs. An identifier is used to give an overall name to the signals (or subscript ranges) contained in the set of parentheses immediately following it, or, if there is no set of parentheses, the identifier is simply a signal name. Integers (or pairs of integers separated by colons) separated by commas indicate allowed ranges on subscripts. The allowed range is 1 to a for the single integer "a", and a to b for the pair a:b. An integer, n, enclosed in parentheses which occurs in a sequence of signal groups merely reserves n as yet unnamed places. This is a distinction not to be overlooked, as Abercrombie (8,5,2) declares 8*5*2 = 80 signals, whereas Fitch ((8), (5), (2)) declares only 8 + 5 + 2 = 15 signals. Finally, the occurrence of "*" followed by an integer, n, is an iterative operator,

and iterates n times the signal group enclosed in parentheses immediately preceding it.

The block components form a listing of the constitutent (previously defined) block names with their terminal lists. The block components, of course, are the elements used to determine the structure of the block being defined.

For clauses may be used to iterate over block components as well as within terminal lists. Iteration takes place on the elements between the DO and matching END. Nesting is permitted.

The block identifer may be repeated, if desired, after the END of a block definition, to improve readability.

3. HOUSING DESCRIPTION

The housing description consists of a sequence of array definitions, each of which may be preceded by a title or drawing statement. There are two types of packaging modules: the connector and the more complex cell. Accordingly, an array definition is either a connector description, or a cell description, or possibly an equivalence statement if it is desirable to give a previously defined array a different name.

In the simplest case, a connector description may be used to describe a two-dimensional array of pins. The generalization of this is to allow a connector to be described in terms of previously defined connectors. As in the case of logic blocks, a connector description must include a connector identifier if the connector is to be used in the definition of higher order arrays. The main body of the connector description is a sequence of array components, which determines the basic make-up of the connector being described.

The purpose of the sequence of array components is twofold: to describe the array in terms of distances between its elements both horizontally and vertically; and to define a nomenclature wherewith the elements can in fact be named. The array vector lists the array elements for a horizontal row of the array. Each array element, however, may actually represent more than one physical element of the eventual array row. The fixed point constant which may occur between array

vector definitions correspond to the vertical spacing between successive rows of the array. The fixed point constant which may occur between array nomenclature expressions corresponds to the horizontal distance between successive elements in the array vector. These distances remain in force until a new spacing definition occurs. If a connector array is made up of connectors other than pins, the connector to be used as a particular element is specified by an array identifier labelling that element. Like distances, array identifiers remain in force until a new identifier is encountered.

A FOR clause may be used to modify the integer expression part of an array nomenclature expression. FOR clauses may iterate sequences of array elements as well as sequences of array components. As with block components, possibilities for nesting FOR clauses are quite broad.

As array nomenclature expressions consist of any alphanumeric character string with optional additive integer expressions, a rule of addition (subtraction) must be defined by any particular implementation of WEB to give definite meaning to an expression such as A5B + 17 which might arise in the iteration of a FOR clause. One rule of addition which would seem to have widespread application would involve defining an alphabetic sequence (eliminating undesirable characters, e.g., I,O) and a numeric sequence. A character position in the nomenclature would be either alphabetic or numeric as defined by the augend. When the end of a sequence is reached, a carry is propagated and the beginning

of the sequence is returned to for that position; in other words, radix addition with a radix scheme based on alphabetic and numeric characters. Further rules would be specified for overflow, underflow, etc.

The cell description may be more complex than the connector description in that cells may possess two types of connectors: external connectors and internal connectors. As usual, a cell must be named to be used in the construction of higher order cells or connectors. The connector identifier specifies the optional external connector of a cell. It is this connector which is used by higher order structures using the cell in their construction. In addition to the external connector, the internal connector of a cell is specified by means of a sequence of array components. A part of this internal connector must be identified with the external connector by having the same nomenclature. The remainder of the internal connector is made up of other connectors corresponding to external connectors of yet more cells, or connectors corresponding to those of the modules (i.e., pluggable units). The distinction between external and internal connectors of a cell is thus quite similar to that of global and local signal names of the logic description.

It should be observed that connector and cell descriptions may be used in any order to build a complex structure. That is, although the cell is more complex than the connector, it is not of higher order. When a pin of a complex structure is to be named, probably the simplest name is the concatenation of the nomenclatures of the connectors, of progressively lower orders, until the pin nomenclature has been uniquely identified.

By way of example, a mother board containing integrated circuit connectors (not wired to those on another board except through the use of an intermediate connector) is a cell. The Illiac III backpanel is a second order connector made up of first order connectors into which boards are inserted. Cabling between backpanels can be treated by making a backpanel an internal connector and a cable connector (or set of cable connectors) an external connector of a cell. At a higher order of description, the IBM System 360 models 50 and 75 in the ASP configuration could be described by making the model 50 and model 75 internal connectors of different cells, while describing the data paths between them in terms of external connectors of these cells.

4. MODULE DESCRIPTION

The module description essentially consists of a sequence of available modules, each of which may be preceded by a title or drawing statement. The module description is used to describe each module type to be used. Logic units described in the logic description are to be assigned to the units of the module. Each such module unit has a pin list, the elements of which correspond in order to the elements of the terminal list of the logic unit which are to be assigned to it. The pin list consists of the nomenclature of the pins as given by the first order connector to which the module may be assigned.

FOR clauses may be used for iteration within pin lists as well as to provide iteration over a sequence of modules, modifying the pin nomenclature at each iteration.

5. ASSIGNMENT DESCRIPTION

The assignment description consists of a module assignment and a unit assignment.

The module assignment is used to assign previously defined modules to first order connectors in the housing. Each assign component consists of the name of a module followed by the list of first order connectors to which the module is to be assigned. The first order connectors are named by the concatenated nomenclature from the highest order connector, in decreasing order, to the position which the first order connector occupies in the second order connector. (Concatenating the pin name of the module description to this connector name yields a complete pin name as indicated in section 3.) FOR clauses may be used to modify connector nomenclature expressions to assign a module or set of modules in any regular pattern.

The unit assignment is used to assign the units of the logic description to the housing, where they are matched with the units of the modules assigned to the housing. As the logic description is expanded, each logic unit description is assigned an index number by which it is referred for the unit assignment. The array nomenclature expression in the fill component refers to the nomenclature of a first order connector to which the unit is to be assigned.

Since the module assigned to a connector will in general have more than one unit, the unit's module position expression is necessary to determine which unit of the module is to correspond to the logic unit being assigned. The module position of a unit refers to the sequential numbers given in the order of occurrence to each of the modules, and hence, module units.

6. SYNTAX OF THE WEB LANGUAGE

Certain syntactical symbols not normally present in BNF are used in the syntax given here. The symbols essentially abbreviate common groupings of BNF statements, and the transformation between these symbols and regular BNF is obvious. The symbols and their interpretation are as follows:

[ ]	entry enclosed is optional
⌊ ⌋	entry enclosed is optional or may be repeated as desired. In terms of Kleene's star, ⌊ A ⌋ = A*
‖ ‖	entry enclosed may be repeated as desired. In terms of Kleene's star, ‖ A ‖ = AA*
{A|B}	exactly one of the symbols A and B may occur

<WEB input> ← [JOB, <Comment>] LOGIC ; <Logic description> HOUSING ; <Housing description>
PLUGGABLE UNITS ; <Module description> ASSIGNMENT ; <Assignment description>

<Logic description> ← ‖ [<Title statement>] [<Drawing statement>] <Block definition> ‖

<Title statement> ← TITLE , <Comment> ;

<Comment> ← / * <Any alphanumeric string not containing * / > * /

<Drawing statement> ← DRAWING, <Drawing name> ;

<Drawing name> ← <Identifier>

<Block definition> ← <Equivalence statement> | <Unit description>
| <Block description>

<Equivalence statement> ← <Identifier> : EQU , <Identifier> ;

<Unit description> ← <Unit identifier> : UNIT , <Terminal list> ;
[<Signal group declaration>]

<Block description> ← { BLOCK | <Block identifier> : BLOCK [, <Terminal list>] } ;
[<Signal group declaration>] ‖ <Block component> ‖
END [<Block identifier>] ;

<Unit identifier> ← <Identifier>

<Block identifier> ← <Identifier>

<Terminal list> ← <Signal element> | , <Signal element> | [<Comment>]

<Signal element> ← <Signal name expression>
| <FOR clause> <Signal element>
| , <Signal element> | END

<Signal name expression> ← <Identifier>
| . <Identifier> | [(<Subscript>
| , <Subscript> |)]

<Subscript> ← <Integer expression>
[: <Integer expression>]

<Signal group declaration> ← DECLARE <Signal group> ⊥ , <Signal group> ⊥ ;

<Signal group> ← <Identifier> [(<Integer> [: <Integer>]
⊥ , <Integer> [: <Integer>] ⊥)]
| (<Integer>)
| (<Signal group>) * <Integer>
| [<Identifier>] (<Signal group>
| , <Signal group> ⊥)

<Block component> ← <Block name> , <Terminal list> ;
| <FOR clause> ‖ <Block component> ‖ END ;

<Block name> ← <Unit identifier> | <Block identifier>

<Housing description> ← ‖ [<Title statement>] [<Drawing statement>] <Array definition> ‖

<Array definition> ← <Equivalence statement> | <Connector description>
| <Cell description>

<Connector description> ← [<Connector identifier> :] PINS ;

‖ <Array component> ‖ END [<Connector identifier>] ;

<Connector identifier> ← <Identifier>

<Array component> ← <Array vector> ; [/ <Fixed point constant> / ;]

| <FOR clause> ‖ <Array component> ‖ END ;

<Array vector> ← <Array element> ⊥ , <Array element> ⊥

<Array element> ← [<Array identifier> :] <Array nomenclature expression>

[, / <Fixed point constant> / ,]

| <FOR clause> <Array element>

⊥ , <Array element> ⊥ END

<Array identifier> ← <Connector identifier> | <Cell identifier>

<Array nomenclature expression> ← <Alphanumeric character string>

[{ + | - } <Integer expression>]

<Cell identifier> ← <Identifier>

<Cell description> ← [<Cell identifier> :] CELL [, <Connector identifier>] ;
‖ <Array component> ‖ END [<Cell identifier>] ;

<Module description> → ‖ [<Title statement>] [<Drawing statement>] <Module name> : MODULE ;
‖ <Module component> ‖ END [<Module name>] ; ‖

<Module name> ← <Identifier>

<Module component> ← <Unit identifier> , <Pin list> ;
| <FOR clause> ‖ <Module component> ‖ END;

<Unit identifier> ← <Identifier>

<Pin list> ← <Pin element> ⊥ , <Pin element> ⊥

<Pin element> ← <Array nomenclature expression>
| <FOR clause> <Pin list> END

<Assignment description> ← [<Title statement>] [<Drawing statement>] <Module assignment>

[<Title statement>] [<Drawing statement>] <Unit assignment>

<Module assignment> ← ASSIGN ; ‖ [<Title statement>] [<Drawing statement>]

<Assign component> ‖ END ;

<Assign component> ← <Module name> , <Connector list> ;

| <FOR clause> ‖ <Assign component> ‖ END ;

<Connector list> ← <Connector element> ⊥ , <Connector element> ⊥

<Unit assignment> ← FILL ; ‖ [<Title statement>] [<Drawing statement>] <Fill component> ‖ END ;

<Fill component> ← <Index number> , <Array nomenclature expression> , <Unit's module position> ;

| <FOR clause> ‖ <Fill component> ‖ END ;

<Index number> ← <Integer expression>

<Unit's module position> ← <Integer expression>

<FOR clause> ← FOR <Identifier> : = <Integer expression>

[STEP <Integer expression>] UNTIL <Integer expression> DO

UNIVERSITY OF ILLINOIS-URBANA
3 0112 039801896